Slowly Changing Dimensions Specification a Relational Algebra Approach

Vasco Santos[1] and Orlando Belo[2]
[1] CIICESI, School of Management and Technology, Porto Polytechnic, Felgueiras, Portugal
Email: vsantos@estgf.ipp.pt
[2] Algoritmi R&D Centre, Universidade do Minho, Braga, Portugal
Email: obelo@di.uminho.pt

Abstract **— In real-world applications, data warehouses usually incorporate some dimension tables in their schemas that have the ability to change over time, reflecting modifications occurred in the respective information sources, involving some of their attributes. These dimension tables are commonly recognized as Slowly Changing Dimensions. Dealing with them is one of the most regular activities of a data warehouse's designer. It is crucial to ensure that the dimension's attributes changes be correctly reflected without affect the data warehouse consistency. For years, researchers and technicians have assumed several Slowly Changing Dimensions strategies using them as the most usual ways to reflect changing and keep historical data. In this paper we will discuss Slowly Changing Dimensions in general terms, presenting their main characteristics and problems, and suggesting a generic formal specification based on Relational Algebra to cover all their known types, with particular emphasis to type 4.**

Index Terms **— Data Warehouse, ETL processes, Slowly Changing Dimensions, Relational Algebra**

I. Introduction

Nowadays, *Data Warehousing Systems* (DWS) [1, 2] play a major role in most of the decision support activities of any company. Decision makers created the habit of using the high-specialized data structures that data warehouses usually maintain materializing their most essential multidimensional perspectives for business analysis and decision. The added value that data warehouses provide to them today is enormous, and besides supporting their information needs they also enrich their common knowledge by bringing new data from very specialized outsourced repositories. Companies recognize today the relevance of a DWS in their daily activities as one of the most important tools they have to support decision-making. From the early beginnings, DWS presented to analysts and designers significant quests. Their impact in a DWS design, and late in their own exploitation, can be faced and interpret from different points of views each of them corresponding to subjective interpretations about their application contexts, functionalities, or business processes, varying from analyst to analyst. Ranging from changing data capture to intensive data loading tasks, it is not difficult to find an operational or a functional requisite that challenge us with some new design and implementation problem. However, perhaps one of the most important quests is to ensure consistency (and correctness) in the data warehouse's structures, especially at dimension table's level. As we know, dimension tables are used to provide subject oriented information by providing data elements to filter, aggregate or describe facts in a data warehouse. Thus, it is quite important that dimensions remain consistent accordingly in some data warehouse's time frame even when some values of their attributes change over time. When this occurs, dimension tables are classified as *Slowly Changing Dimensions* (SCD). In this paper we will discuss SCD strategies in general terms, presenting their main characteristics and problems, and suggesting a global formal specification based on Relational Algebra covering all known SCD treatment variations. In section 2 we will study the SCD dilemma and the most common strategies to deal with them. After a brief discussion, in section 3, we present the Relational Algebra specification that we developed for SCDs, covering especially the case where keep history is a priority. Finally, we present some remarks and conclusions and present some future research lines.

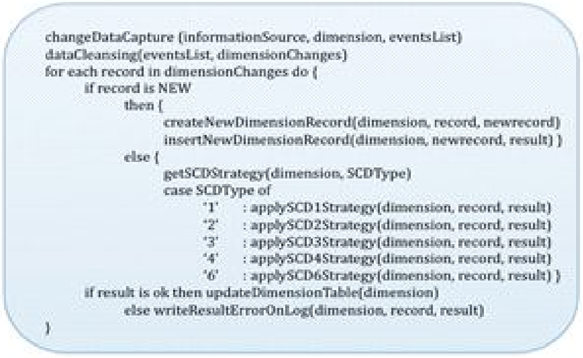
II. Slowly Changing Dimensions

A. *The Traditional Types*

SCD [3] are not a new concept. Often, during the dimensional modeling phase of a data warehouse the term appears frequently, referring to dimension tables that have attributes whose values could change over time. Data warehouse designers sometimes select and integrate attributes in dimension tables that must have special populating procedures to ensure that their values keep consistency as time passes, ensuring that the data warehouse, and in particular the schema where they are integrated, keeps an integrated state in all time frames considered in its own structure [4]. Even requiring such special procedures, SCD are very important to keep up-to-date dimension properties in a data warehouse - an address of a customer or supplier, a production standard cost of a product, a employee's salary, just to name a few -, guaranteeing that facts have an up-to-date dimensional support. Special oriented *Extract, Transform and Load* (ETL) processes [5-7] are responsible to maintain SCD, acting accordingly to the updating strategy previously defined. The way they are implemented followed all the dimensional modeling requisites established for every dimension table classified as SCD; the same occurs if the dimension was classified as *Rapidly Changing Dimension* (RCD); its treatment is quite similar to SCD not differing in any relevant aspect, unless the form it varies in time. In a simplified manner, SCD are treated by ETL processes as is shown in Fig. 1.

Usually, every SCD process starts with a changing event occurrence in a specific information source (one or more) that provides the data to feed the dimension table. Usually, a *Change Data Capture* (CDC) process detects the update event and records it accordingly to some predefined SCD requisites - temporal labels, metadata format, operation markers, operational systems application or user, etc. After, in a DWS staging area the remaining part of the SCD process is triggered and the SCD strategy applied. Although we always speak in terms of process, SCD strategies are applied at the attribute level, for a particular dimension table. For each dimension table, DWS designers use to define what kind of updates attributes will be modified over time. They must prevent all the possible cases of attribute changing, once it is not very recommendable to change dimension table structures when the DWS is already in production. Over a same SCD we could have to apply distinct updating strategies for a single record. Frequently, this involves different processes for different SCD strategies. Based on the most traditional approaches founded in the literature to deal with SCD we can identify the following types [2, 3, 8]:

- *Type 0* - an approach that contradicts everything that a SCD stands for; however, just to mention, when someone refer that a SCD is type 0 means that any change that may occur related to some dimension's attribute will be disregarded.
- *Type 1* - whenever occurs a change in an attribute classified as type 1, the change must be reflected in the dimension's attribute overwriting its previous value; no kind of history will be kept about the values changing.
- *Type 2* - this is the first SCD strategy that allows keeping historical data about changes in dimensions' attributes; every change is recorded as a new dimension record keeping the old one for historical reasons; all the records about a same entity are connected through conceptual pointers, which allows to navigate among all the changes of a specific entity.
- *Type 3* - it allows also to maintain historical data, but only to some level of depth, which is the number of times that we must replicate the definition of the attribute for which we want to keep history; for instance, if we want to keep the last three addresses of a customer (depth 3), we must defined three attributes with the same definition to receive the last three address values.
- *Type 4* - this type considers the use of an auxiliary table especially oriented to keep the changes that dimension's attributes suffer over time, while the dimension table maintain the most recent records; this simplify a lot all the processes required to · update the values of SCD attributes and keeps simple all data browsing tasks over the dimension.
- *Type 6* - basically, this updating strategy is a combination of the techniques used in SCD types 1, 2 and 3, with the objective to usufruct from the advantages of each one of them [8, 9].



```
changeDataCapture (informationSource, dimension, eventsList)
dataCleansing(eventsList, dimensionChanges)
for each record in dimensionChanges do {
        if record is NEW
                then {
                        createNewDimensionRecord(dimension, record, newrecord)
                        insertNewDimensionRecord(dimension, newrecord, result) }
                else {
                        getSCDStrategy(dimension, SCDType)
                        case SCDType of
                                '1'     : applySCD1Strategy(dimension, record, result)
                                '2'     : applySCD2Strategy(dimension, record, result)
                                '3'     : applySCD3Strategy(dimension, record, result)
                                '4'     : applySCD4Strategy(dimension, record, result)
                                '6'     : applySCD6Strategy(dimension, record, result) }
        if result is ok then updateDimensionTable(dimension)
                else writeResultErrorOnLog(dimension, record, result)
}
```

Figure 1. A generic process to deal with SCD

B. Keep or not to Keep History

To keep or not to keep a dimension's table history, in our opinion, is the unique SCD question, and not how to deal with the updating history of the dimensions. In our point of view, when we are speaking about keeping history in dimension tables, all types of SCD can be implemented quite effectively using only SCD type 4. As we will show in the next section, the other types don't have a strong reason to exist. At least right now, because the high technological level that we have today in DWS solves perfectly any kind of SCD case simply and effectively, as well it could improve in most cases the performance of history updating. Let's see why we said this. If we decide that there is no need to keep history in a particular dimension, this dimension table can be designed as static and eventual updates to correct an error or to update some attribute's value, could be easily performed appealing to some kind of rewriting mechanism. Nothing more is necessary to do. However, if we say that we need to keep the history of a dimension table, then we suggest that we create a second table - let's say a dimension's history table (Fig. 2) (like it happens in SCD type 4) - as the repository to everything that we want to keep about dimension's past occurrences. The structure of the second table can be defined in two perspectives:

- *pessimist*, where the designer feels that there is a possibility that any one of the dimension's attributes changes over time; in this case we recommend that the history table structure follows the structure of the dimension table, unless it is a huge dimension, having a huge number of attributes, for obvious reasons;
- *optimist*, in which the designer is convict that he considered all the important aspects during requirement analysis phase, and he knows, for sure, what are the attributes that change over time and the way they do that.

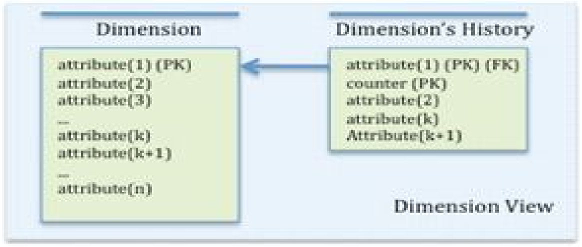


Figure 2. A view of a dimension table integrating an auxiliary history table

Independently from wanting to keep history of the entire dimension's record (as it happens in conventional SCD type 2), or only of one or two attributes (an SCD type 3 case) we always use the history table to receive updates. In the first case, we copy the current dimension's record to the history table and then we rewrite it with the new information. To do this operation we don't need to use any kind of surrogate keys (even though, as we know, the use of surrogate keys in some cases brings us enormous advantages in terms of storage and performance and must be used). It only requires the existence of a new attribute in the history table, basically a counter, which corresponds to the number of times that a specific record was updated. It will be very useful when we need to order history occurrences. In order to enrich a little bit the history table, we can add some more attributes related to the date and time of the update occurrence, information source involved or the process that provoked it. For the second case, we will act in the same way. It is true that we will repeat some attribute's values from new record to new record, introducing redundancy in the history table. But if we take into consideration that every time an updating event occurs the dimension table remains stable, in both cases, able to be browsed or joined at any time without the need of any kind of views (or other filtering mechanisms) it is a great gain in DWS usage, data exploitation, and performance. Every time we think in the great confusion that a SCD type 2 or 3, especially SCD type 2, provokes in the structure of the dimension table, and the difficulties that emerge from that every time we want to cross some data, we largely prefer to put some redundancy in an (auxiliary) history table and keep dimension table ready for querying. However, due to its nature, changes in SCD tables are not often. So, the redundancy of a dimension's history table will not reach "drastic" levels in real world application, even for cases where SCD change rapidly. Of course, we assume that any DWS designer is able to do, for a particular application context, the right balance between the two extreme options: pessimist or optimist. Sometimes that will be quite difficult due to the community of decision makers and the inability that sometimes they have to communicate their own business decision-support directives. However, at least, now he only needs to decide to keep or not to keep dimension's history. In the next section we will present a Relation Algebra specification for SCD type 4, showing a formal way to describe its process as the way we see it. Additionally, only as a term for comparison, we also present the specification of all the others conventional SCD types.

C. Yet Another Approach

Recently, there has been a new type used to classify a complex SCD. If a dimension contains attributes that behave differently, in the context of history preservation, like for instance type 1, 2 and 3, it is common to identify it as a SCD type 6. Classifying a SCD as a Type X is not correct, types of SCD are applied to attributes, so in general we could have dimensions with attributes that are type 1, other attributes that are type 2 or type 3. Since it is more common to find dimensions that have more than one of the types analyzed previously, dealing with these changes has just become more complicated. A simple example that demonstrates the increased complexity is when a dimension that contains several types of attributes gets updated. Suppose that a type 1 attribute is updated, since there is no regard over historical values a simple update should be made to the attribute. The problem arises if that dimension also contains a type 2 attribute that has already been updated several times. Those updates generated new records with new surrogate keys. In order to maintain the dimension in a valid state the update to the type 1 attribute should also be made to all other records that share the same business key. This consequence overburdens the already complex task of designing how to deal with SCD and consequently the Relational Algebra specification. After comparing all the approaches to deal with SCD it is our view that there is no need to type SCD since a single approach handles all problems with increased benefits and low complexity [10].

III. Using Relational Algebra for the Specification of SCD

Let's assume that the ETL process extracts all the data existent in a source system to load into a specific dimension. We shall call the table that stores that data *source_data*. The definition of a table, or relation, as first presented in [11] and described in many subsequent articles and textbooks, is a set n-tuples with each element of the tuple corresponding a specific domain. Thus, we define the relation *source_data* as a list of attributes that comprise a business key and a finite set of additional attributes (1). The business key is normally identified as a primary key and may be a single or set of attributes.

$$source\_data = \langle A_1, \dots, A_n, B_1, \dots, B_m \rangle \quad (1)$$

in which $A_1, \dots, A_n$ are the attributes that belong to the Business Key and $n >= 1$, and $B_1, \dots, B_m$ are additional data attributes needed in the dimension and $m >= 1$.

A. SCD Type 1

When an attribute is identified as a SCD type 1 attribute, the methodology to deal with the changes is simple, the attribute is simply updated. Since the old attribute value is lost, all future analysis refer to the new value, therefore history is not retained. Typically, the difference between the *dimension* structure (2) and the *source_data* relation is solely the presence of an attribute, surrogate key, which identifies the tuple of the dimension.

$$dimension = \langle SK, A_1, \dots, A_n, B_1, \dots, B_m \rangle \quad (2)$$

in which *SK* is the surrogate key, normally a natural number. Let's now consider that the type 1 attribute is one of the $B_1, \dots, B_m$ attributes. In order to maintain the dimension table with conformed data, a series of operations need to be executed. We assume that the ETL process loads the *source_data* relation with all significant data present in the source systems, i.e., a generic approach. Given this approach we first need to remove from *source_data* the unchanged tuples, i.e., retain the new and changed tuples only (3). This step is used in every type of SCD, thus a more optimized approach could be loading the *source_data* relation with just new or updated tuples from source systems.

$$NC\_data \leftarrow source\_data - \pi_{A_1,\dots,A_n,B_1,\dots,B_m}(dimension) \quad (3)$$

The *NC_data* relation now needs to be divided in two relations, separating the new tuples, which will be added to the dimension after the attribution of a surrogate key, from the changed tuples that need a more complex set of operations. In order to identify changed dimension data we need to know their business key. An intersect operation over *dimension* and *NC_data* will provide us with the needed information (4). We shall name it *ChangedBK* to identify a relation that contains the business keys of changed tuples. It is now possible to identify dimension data that has been updated (5) and therefore remove it (6) in order to substitute it with the new updated data (7)(8).

$$ChangedBK \leftarrow \pi_{A_1,\dots,A_n}(dimension) \cap \pi_{A_1,\dots,A_n}(NC\_data) \quad (4)$$

$$Old\_data \leftarrow ChangedBK \infty\, dimension \quad (5)$$

$$dimension \leftarrow dimension - Old\_data \quad (6)$$

$$Changed\_data \leftarrow \pi_{SK,A_1,\dots,A_n}(Old\_data) \infty\, NC\_data \quad (7)$$

$$dimension \leftarrow dimension \cup Changed\_data \quad (8)$$

After dealing with updated data, the new data must also be dealt with (9). The addition of new data requires the ETL process to assign a new surrogate key for every new tuple. This process is not the primary focus of this paper and will be studied in future work. For now we assume that through the use of the extend operator [12] a new attribute (SK) is added to *New_data* and filled with valid values (10). After that process the addition of the new values to dimension is done through the use of the union operator (11).

$$New\_data \leftarrow NC\_data - (ChangedBK \infty\, NC\_data) \quad (9)$$

$$New\_data \leftarrow \varepsilon_{[SK=new\_surrogate()]}(New\_data) \quad (10)$$

$$dimension \leftarrow dimension \cup New\_data \quad (11)$$

B. SCD Type 2

When an attribute is identified as a SCD type 2 attribute, the methodology to deal with the changes is complex. Since history is preserved in the dimension relation, additional attributes are needed to identify the timeframe in which the tuples are valid. Therefore the structure of the dimension relation (12) is somewhat different than in SCD type 1.

$$dimension = \langle SK, A_1, \dots, A_n, B_1, \dots, B_m, DateFrom, DateTo, current \rangle \quad (12)$$

where *DateFrom* and *DateTo* are two *Data* type attributes, *current* attribute is typically a bit attribute that identifies if tuple is actual or not. Let's consider that the type 2 attribute is one of the $B_1, \dots, B_m$ attributes. In order to process correctly the changes in the type 2 attribute, we first need to separate current from historical data (13) since updates will force current data to become history by changing the *current* and *DateTo* attribute.

$$dimension\_current \leftarrow \sigma_{current=1}(dimension) \quad (13)$$

Now, we use *dimension_current* data to identify new and changed data from source systems (14). Equations (15) and (16) are almost identical to (4) and (5) of type 1 approach, since the objective is to update changed data. To complete the expiring process we add the changed data with *DateTo* attribute set to the expiring date, normally the date of the day the ETL process is running, and *current* attribute set to 0 which means expired data (17).

$$NC\_data \leftarrow source\_data - \pi_{A_1,\dots,A_n,B_1,\dots,B_m}(dimension\_current) \quad (14)$$

$$ChangedBK \leftarrow \pi_{A_1,\dots,A_n}(dimension\_current) \cap \pi_{A_1,\dots,A_n}(NC\_data) \quad (15)$$

$$Old\_data \leftarrow ChangedBK \infty\, dimension\_current \quad (16)$$

$$dimension \leftarrow dimension - Old\_data \quad (6)$$

$$dimension \leftarrow dimension \cup \pi_{SK,A_1,\dots,A_n,B_1,\dots,B_m,DateFrom,today(),0}(Old\_data) \quad (17)$$

To finalize the type 2 methodology, the new and changed data is added to the dimension with appropriate temporal data and a new surrogate key (18) (19). Note that (18) treats both changed data and new data in the same way, i.e., generates a new surrogate key and timestamps the validity of the tuples to be added to the dimension.

$$NC\_data \leftarrow \varepsilon_{[SK=new\_surrogate(),DateFrom=today(),DateTo=31/12/9999,current=1]}(NC\_data) \quad (18)$$

$$dimension \leftarrow dimension \cup NC\_data \quad (19)$$

C. SCD Type 3

As a compromise to the disadvantages present in type 2 approach, type 3 is able to store historical changes to the degree needed. This is accomplished by the used of additional attributes that store previous values. Typically only one level is used, but it is possible, but not advisable, to use more. In order to process properly this approach we need to correctly identify the attribute in the *source_data* relation (20). The structure of the dimension relation therefore reflects this approach (21).

$$source\_data = \langle A_1, \dots, A_n, B_1, \dots, B_i, \dots, B_m \rangle \quad (20)$$

$$dimension = \langle SK, A_1, \dots, A_n, B_1, \dots, B_i, \dots, B_m, B_{previous} \rangle \quad (21)$$

in which B_i is the type 3 attributes and $1 <= i <= m$, and $B_{previous}$ is the attribute that will hold the previous B_i value. In order to identify new and changed data coming from source systems, we use the steps (3) through (6) defined in type 1 approach.

$$NC\_data \leftarrow source\_data - \pi_{A_1,\dots,A_n,B_1,\dots,B_m}(dimension) \quad (3)$$

$$ChangedBK \leftarrow \pi_{A_1,\dots,A_n}(dimension) \cap \pi_{A_1,\dots,A_n}(NC\_data) \quad (4)$$

$$Old\_data \leftarrow ChangedBK \infty\, dimension \quad (5)$$

$$dimension \leftarrow dimension - Old\_data \quad (6)$$

This approach requires that whenever a change occurs, the previous value is stored in attribute $B_{previous}$ and the current value of the attribute is stored in B_i. This is accomplished through the use of the Relational Algebra rename operator (22). Afterwards the data is added to the dimension (8).

$$Changed\_data \leftarrow \pi_{SK, A_1,\ldots,A_n, B_{previous}}(\rho_{B_{previous}/B_i}(Old\_data)) \infty NC\_data \qquad (22)$$

$$dimension \leftarrow dimension \cup Changed\_data \qquad (8)$$

Dealing with new data in this approach is similar to type 1 approach, the only difference is the value stored in $B_{previous}$ that must be set to *NULL*, since there are no previous values in new data (23).

$$New\_data \leftarrow NC\_data - (ChangedBK \infty NC\_data) \qquad (9)$$

$$New\_data \leftarrow \varepsilon_{[SK=new\_surrogate(),\ B_{previous}=NULL]}(New\_data) \qquad (23)$$

$$dimension \leftarrow dimension \cup New\_data \qquad (11)$$

D. SCD Type 4

In this approach, a *dimension* is represented by two relations, one that stores current data, that we shall from this point forward call *dimension_current*, and the other one stores the historical changes that became expired, that we identify as *dimension_history*. We assume for this approach that the source data relation has the same structure as the precious approaches (1). The *Dimension* relation is formed by two relations, *dimension_current* and *dimension_history*, with the same structure (24). The difference between the structure of these two relations and the *source_data* relation is solely the presence of an attribute, surrogate key, that identifies the tuple of the dimension and a date attribute identifying the temporal validity of the tuple. The use of the surrogate key is not mandatory but helps, for instance, in cases where *Business Key* is a composite key. The attribute *Date* timestamps the beginning of the validity of the tuple helping in cases where some historical order is needed.

$$dimension\_current = \langle SK, A_1, \ldots, A_n, B_1, \ldots, B_m, Date \rangle \qquad (24)$$

The approach to determine new and changed data and the operations needed to update correctly the dimension's relations is somewhat similar to the previous approaches (25), the only difference being the fact that we use separate relations to store current and historical data. In (26)(27) we identify changed data that needs to be transferred from *dimension_current* to *dimension_history* (28)(29).

$$NC\_data \leftarrow source\_data - \pi_{A_1,\ldots,A_n, B_1,\ldots,B_m}(dimension\_current) \qquad (25)$$

$$ChangedBK \leftarrow \pi_{A_1,\ldots,A_n}(dimension\_current) \cap \pi_{A_1,\ldots,A_n}(NC\_data) \qquad (26)$$

$$Old\_data \leftarrow ChangedBK \infty dimension\_current \qquad (27)$$

$$dimension\_history \leftarrow dimension\_history \cup Old\_data \qquad (28)$$

$$dimension\_current \leftarrow dimension\_current - Old\_data \qquad (29)$$

To insert changed data in *dimension_current* we use (7) to identify the changed data preserving their surrogate key by accessing the temporary relation *Old_data* defined in (27). In (30) we add an additional attribute called *Date* that will provide us with a timestamp day of the changed data through the use of a system-defined function *today()*. Finally, the insert operation is accomplished through the use of the union operator in (31).

$$Changed\_data \leftarrow \pi_{SK, A_1,\ldots,A_n}(Old\_data) \infty NC\_data \qquad (7)$$

$$Changed\_data \leftarrow \varepsilon_{[Date=today()]}(Changed\_data) \qquad (30)$$

$$dimension\_current \leftarrow dimension\_current \cup Changed\_data \qquad (31)$$

After determining the business keys of the changed data in (26) we can use that information to identify new data (9) and insert it in *dimension_current*. The addition of new data requires the ETL process to assign a new surrogate key for every new tuple. After that process the new tuples are added to *dimension_current* (32).

$$New\_data \leftarrow NC\_data - (ChangedBK \infty NC\_data) \qquad (9)$$

$$New\_data \leftarrow \varepsilon_{[SK=new\_surrogate()]}(New\_data) \qquad (10)$$

$$dimension\_current \leftarrow dimension\_current \cup New\_data \qquad (32)$$

Conclusions

One of the most challenging tasks a DW designer faces is to design an ETL process that will perform the tasks needed in the time available with success. Designing how to deal with SCD is of superior importance since it allows (or not) to analyze data with or without historical changes as needed. There are common approaches to deal with this SCD that, in our view, increase, unnecessarily, the complexity of the ETL process. If history is important for future data analysis, SCD Type 4 is a clean answer to all identifiable problems. In this paper we present some reasons that make us believe that there is only one approach viable and therefore no need to type SCD. Nevertheless we present a Relational Algebra specification of all common types of SCD used in DWS and identify common steps shared by all approaches, as a step forward for a more formal specification of ETL processes. In future work we intend to extend this work to the specification of surrogate key generation and surrogate key pipelining.

References

[1] W. H. Inmon, Building the Data Warehouse, 3rd ed.: John Wiley & Sons, 2002.

[2] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, and B. Becker, The Data Warehouse Lifecycle Toolkit - Practical Techniques for Building Data Warehouse and Business Intelligence Systems, Second Edition ed. : Wiley Publishing, Inc., 2008.

[3] R. Kimball and J. Caserta, The Data Warehouse ETL Toolkit - Pratical Techniques for Extracting, Cleaning, Conforming, and

Delivering Data. : Wiley Publishing, Inc., 2004.
[4] D. A. J. Griffin, P. J. L. Griffiths, S. H. Judges, N. A. Campbell, and M. D. Roberts, "Method of Managing Slowly Changing Dimensions," United States of America Patent US 6,847,973 B2, 2005.
[5] A. Simitsis, P. Vassiliadis, and T. Sellis, "Extraction-Transformation-Loading Processes.," in *Encyclopedia of Database Technologies and Applications*, L. C. Rivero, J. H. Doorn, and V. E. Ferraggine, Eds., ed: Idea Group, 2005, pp. 240-245.
[6] A. Simitsis and P. Vassiliadis, "A method for the mapping of conceptual designs to logical blueprints for ETL processes," *Decision Support Systems*, vol. 45, pp. 22-40, 2008.
[7] P. Vassiliadis, A. Simitsis, and E. Baikousi, "A taxonomy of ETL activities," presented at the Proceeding of the ACM twelfth international workshop on Data warehousing and OLAP, Hong Kong, China, 2009.
[8] R. Kimball and M. Ross, *The Data Warehouse Toolkit - The Complete Guide to Dimensional Modeling*, Second ed. : John Wiley & Sons, 2002.
[9] T. Manh Nguyen, A. Min Tjoa, J. Nemec, and M. Windisch, "An approach towards an event-fed solution for slowly changing dimensions in data warehouses with a detailed case study," *Data & Knowledge Engineering*, vol. 63, pp. 26-43, 2007.
[10] V. Santos and O. Belo, "No Need to Type Slowly Changing Dimensions," in *IADIS International Conference Information Systems 2011*, Avila, Spain, 2011, pp. 129—136.
[11] E. F. Codd, "A relational model of data for large shared data banks," *Commununications of the ACM*, vol. 13, pp. 377-387, 1970.
[12] E. Baralis and J. Widom, "An Algebraic Approach to Rule Analysis in Expert Database Systems," in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 475 — 486.